Übung: Stochastic simulations –simple version

SS 2010 Übung zu Struktur und Simulation

28-Jun-10

1. Timeplan

* June 29: start coding

* July 6: finish programming, run calculations

* July 13: hand in report

From *n* students, there should be *n* different reports.

2. Introduction

When we model chemical reactions, we normally use simple differential equations. If we have an irreversible reaction, A+B→C, you might say $\frac{dX_A}{dt} = -kX_A X_B$ where X_i is the concentration of species *i*. If we have a few molecules of type A or B, this may not be a useful model. The concentration of molecules *A* and *B* is so low, that they do not often collide. If you double X_A, you do not always see the reaction rate double. In biology there may be only 10's of copies of a transcription factor in some cells. If the promoter is switched on, a gene is activated, an enzyme synthesised and you see the reaction, A+B→C. Sometimes, the promoter is not able to bind before it is degraded, so the reaction does not take place at all.

In finance, you may have a model for supply and demand and attempt to find the point where they are equal. If you only have a few customers, this may not be a good predictor of the price. A product may or may not be sold depending on whether one of the few buyers is interested on a particular day. In football, one team may be 10 % better than another, but this is not very useful

in predicting the outcome. A football Tor/Goal is an infrequent event. In 1977, Gillespie proposed a method[1] which is useful for simulating the kinetics of low frequency events and football games.

In this Übung, one must write a program to implement part of Gillespie's algorithm. In a later Übung, we run a version of the full algorithm.

2.1.Final program

We will simulate football games between Australia and Brazil. We obviously need to know how many games are won by each team and how many are "unentschieden". Our local betting office (Wettbüro) allows us to place bets during half time. We want to know if the score at half time is a good predictor of the final score.

The invocation of the program should be

```
fussball seed n_game rel_strength
```

where

`seed`	integer random number seed
`n_game`	number of games to be played
`rel_strength`	relative strength. If Australia is 10 % stronger, this would be 1.1. If Australia is 23 % better, this would be 1.23.

Your program should read the command line as given above, so it can be invoked from a script. We want summaries of the games as well as the information about half time scores.

N_{team1}	Number of games won by team 1
N_{team2}	Number of games won by team 2
N_{draw}	Number of unentschieden games

And we also need a table

[1] Gillespie, D.T., J. Phys. Chem. 81, 2340-2361 (1977), "Exact stochastic simulation of coupled chemical reactions".

		Final winner		
		team 1	team 2	unentschieden
winner at half time	team 1			
	team 2			
	unentschieden			

In this table, one can read across the row and see how many times team 1 (or 2) was the winner at half time and then was the winner at the end.
As an example, try running the executable `~/torda/uebung_stochastic/fussball`.

3. Algorithm

Gillespie's basic method was to take steps in time, each of random length. After each step, one asks if a random event has occurred and then updates the concentrations of the reactants. We use two simplifications. We use steps of constant length and we do not have to update reactant concentrations. If we work with very small time steps, we can say the probability of an event happening twice is negligible, so we do not have to account for the possibility of two goals in one minute. To simulate a game, one has to

```
initialise all counters;
for (i = 0; i < num_steps; i++)
    if (makes_goal (team1))
        score[1] += 1;
    if (makes_goal (team2))
        score[2] += 1;
```

To decide whether a goal is scored, you first need the probability of a goal within a minute (`prob`) for that team and a function to make a decision:

```
unsigned
makes_goal (float prob)
{
    double r = drand48();
    if (r < prob) /* goal scored */
        return 1;
    else
        return 0;
}
```

An implementation of a useful program is more complicated.

4. Code

4.1. Coding rules

- You can program in any language you want. It should provide a deterministic uniform random number such as `drand48()` in C or `rand()`in perl, `rand` in ruby or `random()` in python. A working program in fortran77 receives an instant beer. You can ask for help if you write in C or perl.
- Your code must read the command line arguments as given on page 3. If my shell script cannot run your program, it does not count as a functioning program.
- You have to know how to read command line arguments in the language you have picked such as `argc`/`argv[]` in C or `@ARGV` in perl, `ARGV` in ruby, `sys.argv` in python, `args` in java, …
- Your program must compile and run on the machines in the teaching pool.

4.2. Coding advice

- The algorithm is very simple, but you have to be very careful passing results around. Plan to store goals from team1, team2 after the first half and at the end of the game. Plan to store an array corresponding to the table on page 2.
- You will need some arrays and two-dimensional arrays, but we will not need any dynamic memory allocation. We know how many teams we have and all the possible results, so you may hard-code array sizes.
- There is not enough time to write an elaborate program. Accept some limitations:
    - there are only two teams
    - you can hard-code constants such as the basic average number of goals per side, per game (referred to as s_{av} and set at 0.4 below)
    - there are 2 × 45 minute half-games

4.3. Suggested structure

You might like a basic structure and set of functions

- `makes_goal()` for a given probability, return whether or not a goal was scored
- `half_game()` return the number of goals scored by each time after one half game
- `game()` get the number of goals scored by each time in the 1st and 2nd halves

- `many_games()` collect the number of times you have a victory by team1, team2 and unentschieden. More difficult - count the number of times team1 wins when team1 was winning at half time, when team2 was winning at half time and so on
- `main()` Set up score and total matrices. Set the seed for the random number generator. Calculate the probability for each team to score a goal in any minute. Print out the final results.

To calculate probabilities for a team scoring in some minute, say, $p = \frac{rs_{av}}{m} = \frac{r \cdot 0.4}{90}$ where *r* is the relative strength of one side (a number like 1.1 for one team and 1.0 for the other). s_{av} is the average number of goals to be scored by one team in a game. 0.4 is an arbitrary choice, but please use it so you can compare your results to a working program. *m* is the number of minutes per game.

5. Testing

Set *n_games* to a small number and add debugging lines to print out the results from each game and make sure you are passing results correctly back from each function. As in real football, scores should be rather low. Set *n_games* to a large number and set the relative strength to 1.0 and see if you get roughly equal numbers of wins for teams 1 and 2. Remember, you will have lots of results "unentschieden". For a large *n_games*, see if you get similar results to the executable mentioned on page 3.

6. Assignment

1. Write a program which simulates the football games

2. Take the first letter of your first name and the first letter of your family name. Convert them to numbers saying "a"=1, "b"=2, … This will give you two digits *a* and *b*. Set the relative strength of Australia to $1 + a/100 + b/100$ so Franz Beckenbauer would use 1 + 0.06 + 0.02 = 1.08.

Pick two large random numbers to be used as your random number seeds.
Write down the relative strength of the Australian football team.
Write down your two random number seeds.

3. Set *n_games* to 10. Collect results for both random number seeds. Write down the results for both tables as given on page 2.

4. Set *n_games* to 100000. Collect the results as for question 3.

5. You should have seen that the results at half time are a reasonable predictor for the final result. The scores seem to be rather correlated. If you increased the average number of goals per game

per side to a number greater than 0.4, would this correlation become stronger or weaker ? You can test your idea by changing your code or you can argue from a probabilistic basis.
6. In these calculations, time was treated in units of one minute. Which approximation would break down if you used steps of 10 minutes ?
7. Australia is rated as the best team in the world cup with a probability p=0.6 of winning against anyone. If Australia reaches the quarter finals, what is the probability that they will win the championship if it is played as a "knock-out tournament" (Turnier nach der K.O. System) ?

Send answers to `gst_uebung@zbh.uni-hamburg.de`.